CamelOne 2013

June 10-11 2013

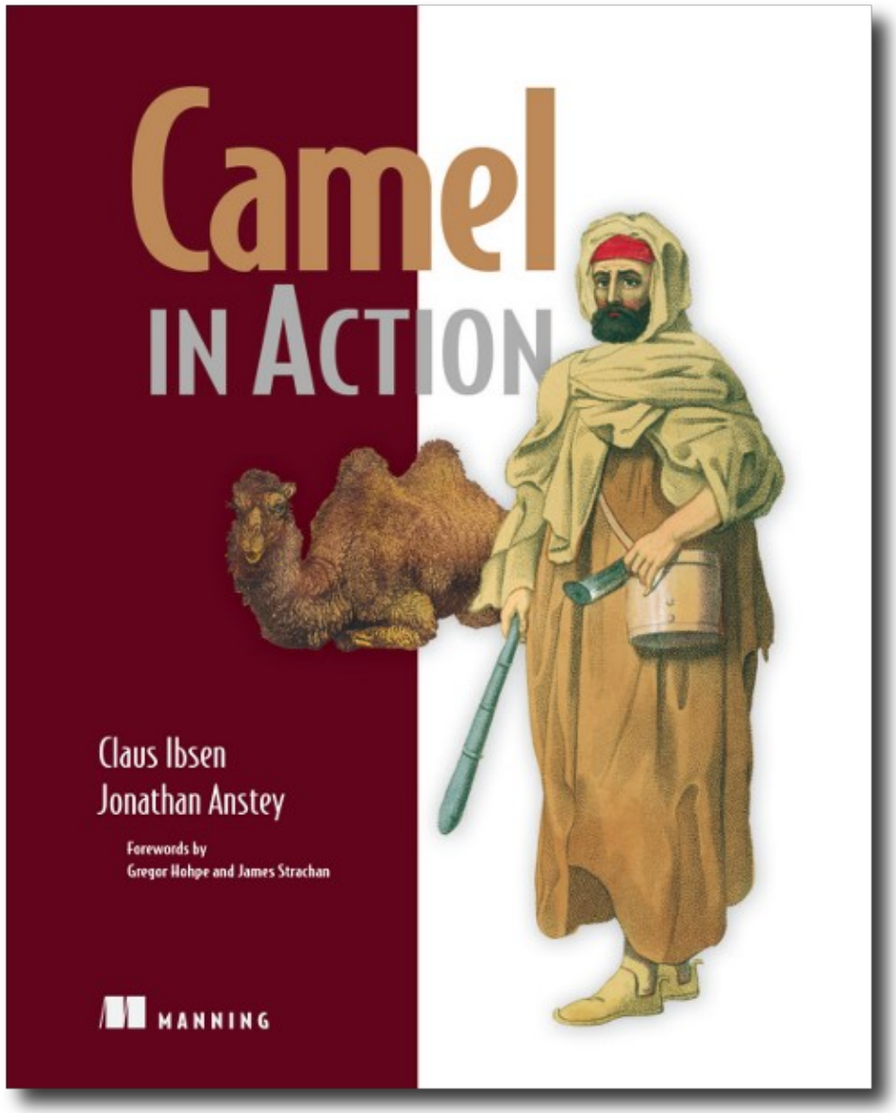
Boston, MA

Enterprise Integration with Apache Camel and ServiceMix

Jonathan Anstey
Principal Engineer, Red Hat

About me

- Principal Engineer at Red Hat
- Apache Camel, ActiveMQ, and ServiceMix committer
- Co-author of Camel in Action
- Twitter: @jon_anstey
- Blog: http://janstey.blogspot.com



Agenda

- What is Apache ServiceMix?
- ServiceMix project update
- Tips for deploying Camel apps into ServiceMix
- Rider Auto Parts example and demo

What is Apache ServiceMix?

CamelOne 2013

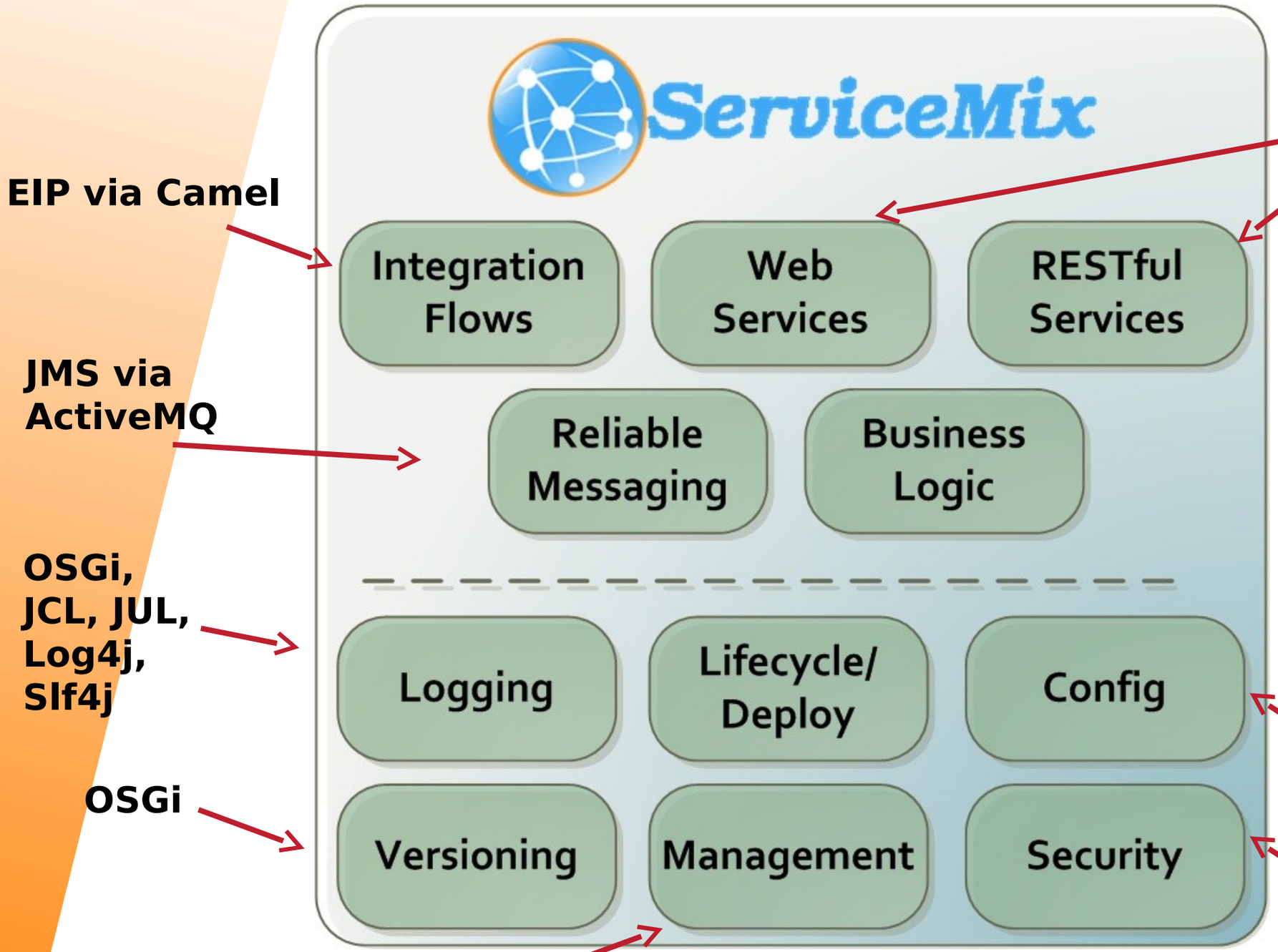
What is ServiceMix?

- Open source container useful for integration and SOA – an ESB.
  - EIP-style integration flows
  - SOAP & REST web services
  - Reliable messaging

What is ServiceMix?

- Support for various cross-functional concerns
  - Logging
  - Lifecycle and deployment
  - Configuration
  - Versioning & Dependency Mgmt
  - Management
  - Security
  - Transactions

The Technologies



ServiceMix 4.5.1

- Released in March
- Updated components
  - Camel 2.10
  - ActiveMQ 5.7
  - CXF 2.6
- Support for Activiti BPM

ServiceMix 5 and beyond

- Hoping to release version 5 this year
- First release that will **NOT** include JBI
- Much tighter build so releases are planned to be more frequent
- Also, Fuse Fabric is in the process of being donated to the Apache Software Foundation under the ServiceMix project

Fuse Fabric

Fuse Fabric

- Fabric is a framework for configuring, provisioning and running Fuse and Apache integration software on any number of machines
- http://fuse.fusesource.org/fabric

Where do I start?

Start quickly with Maven archetypes

- Apache Maven archetypes are project templates
- Use camel-archetype-blueprint to create new blueprint-based Camel route
- Fuse IDE, Eclipse, IntelliJ support this

OSGi-ifying existing project

- Change Maven POM packaging type
  - <packaging>bundle</packaging>
- Use the maven-bundle-plugin to generate OSGi entries in the JAR's MANIFEST

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <extensions>true</extensions>
</plugin>
```

CamelOne

Using what the ESB has to offer

Take advantage of OSGi Config Admin

- The Config Admin service provides an easy way of getting configuration into your bundle

```
<property-placeholder persistent-id="org.fusesource.camel.file"
  xmlns="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.0.0">
  <default-properties>
    <property name="fileEndpoint" value="file:target/placeorder" />
  </default-properties>
</property-placeholder>
```

- You can then use these properties in your routes

```
<camelContext id="rider-auto-file-poller"
  xmlns="http://camel.apache.org/schema/blueprint">
  <route id="file-to-jms">
    <from uri="{{fileEndpoint}}" />
    <to uri="jms:incomingOrders" />
  </route>
</camelContext>
```

Take advantage of OSGi Config Admin

- You can update properties in the command shell or by modifying a properties file.
- Updating the file endpoint at runtime is simple:

```
karaf@root> config:edit org.fusesource.camel.file
karaf@root> config:propset fileEndpoint file:/tmp/my_directory
karaf@root> config:update
karaf@root> osgi:restart 216
```

Reference existing services

- You should reuse existing services rather than rolling your own
- Reference ActiveMQ ConnectionFactory for JMS messaging

```
<reference id="connectionFactory" interface="javax.jms.ConnectionFactory"
  filter="(name=localhost)" />

<bean id="jms" class="org.apache.activemq.camel.component.ActiveMQComponent">
  <property name="connectionFactory" ref="connectionFactory" />
</bean>
```

- Reference Aries TransactionManager for transactions

```
<reference id="transactionManager" interface="javax.transaction.TransactionManager" />
```

Decouple subsystems with JMS

- Decouple sub systems by using JMS queues hosted on ActiveMQ broker

```
<route id="file-to-jms">
  <from uri="{{fileEndpoint}}" />
  <to uri="jms:incomingOrders" />
</route>
```

```
<route id="normalize-message-data">
  <from uri="jms:incomingOrders" />
  <choice>
    <when>
      <simple>${header.CamelFileName} regex '^.*xml$'</simple>
      <unmarshal>
        <jaxb contextPath="org.fusesource.camel.model" />
      </unmarshal>
    </when>
    ...
```

Testing before deploying...

CamelOne 2013

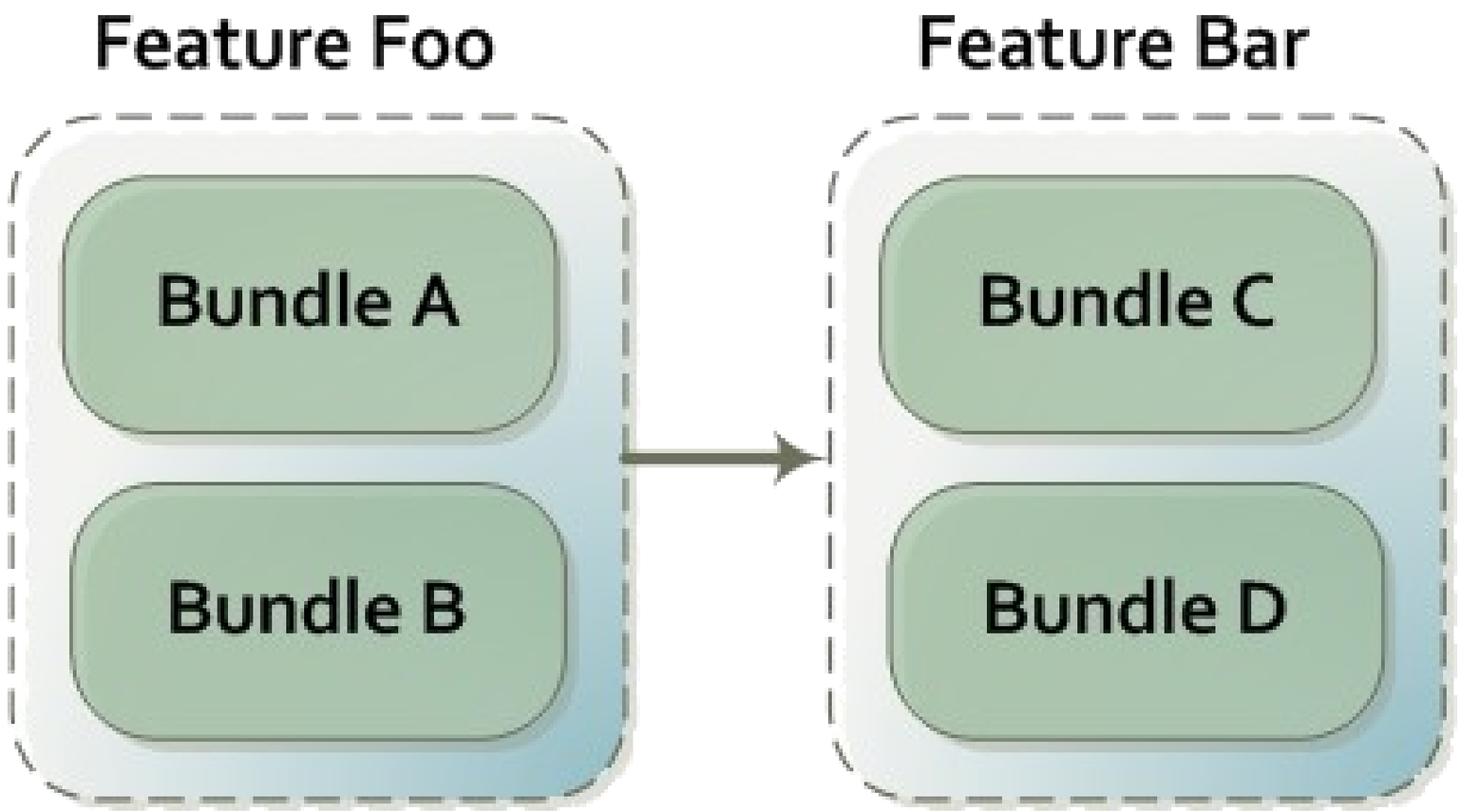
Testing

- Use camel-test-blueprint to define unit tests for your blueprint-based routes
- Use Pax Exam to test routes that require services from the container (like JMS broker, transaction manager, etc)

Grouping bundles...

Deploy with features

- Features group bundles into a logical unit of deployment
- Installing feature "Foo" would install bundles A, B, C and D



Deploy with features

- You should specify existing features in ServiceMix to depend on rather than individual bundles.

```
<features name="rider-auto-osgi"
  xmlns="http://karaf.apache.org/xmlns/features/v1.0.0">
    <feature version="${project.version}" name="rider-auto-osgi">
        <feature>camel-core</feature>
        <feature>camel-blueprint</feature>
        <feature>camel-activemq</feature>
        <feature>camel-jaxb</feature>
        <feature>camel-bindy</feature>
        <feature>camel-cxf</feature>
        <bundle>mvn:org.fusesource.examples/rider-auto-common/${project.version}</bundle>
        <bundle>mvn:org.fusesource.examples/rider-auto-file/${project.version}</bundle>
        <bundle>mvn:org.fusesource.examples/rider-auto-ws/${project.version}</bundle>
        <bundle>mvn:org.fusesource.examples/rider-auto-normalizer/${project.version}</bundle>
        <bundle>mvn:org.fusesource.examples/rider-auto-backend/${project.version}</bundle>
    </feature>
</features>
```

- You can then SSH into ServiceMix and use the features shell to install the feature.

Managing runtime routes using Karaf Camel commands...

Karaf Camel Commands

- Start/stop routes and contexts deployed in ESB
- View route XML and stats
- Many commands available

```
karaf@root> camel:
camel:context-info      camel:context-list      camel:context-start
camel:context-stop      camel:endpoint-list     camel:route-info
camel:route-list        camel:route-resume      camel:route-show
camel:route-start       camel:route-stop        camel:route-suspend
```

CamelOne

Tweaking your ESB...

CamelOne 2013

Deploying only what you need

- You should reduce the boot features to only what you need.
  - featuresBoot property in etc/org.apache.karaf.features.cfg
- Vanilla install of Apache ServiceMix loads over 170 bundles

Making sure you don’t need internet access

- Maven is great for development time as it just downloads from repositories on the Internet.
- In production you should make sure all libraries are available locally
  - May not have Internet access
  - Reduces the risk of failure at deploy time

- Easy way: use the "full" distribution
  - Contains libraries for all features in system directory

Making sure you don't need internet access

- Use the features-maven-plugin to package up all 3rd party dependencies of your application.

```
<plugin>
  <groupId>org.apache.karaf.tooling</groupId>
  <artifactId>features-maven-plugin</artifactId>
  <executions>
    <execution>
      <id>add-features-to-repo</id>
      <phase>generate-resources</phase>
      <goals>
        <goal>add-features-to-repo</goal>
      </goals>
      <configuration>
        <descriptors>
          <descriptor>mvn:org.apache.camel.karaf/apache-camel/${camel-version}/xml/features</descriptor>
          <descriptor>mvn:org.apache.servicemix/apache-servicemix/${servicemix-version}/xml/features</descriptor>
          <descriptor>mvn:org.apache.activemq/activemq-karaf/${activemq-version}/xml/features</descriptor>
          <descriptor>file:${basedir}/target/classes/features.xml</descriptor>
        </descriptors>
        <features>
          <feature>rider-auto-osgi</feature>
        </features>
        <repository>target/repo</repository>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Making sure you don’t need internet access

- These dependencies should then be made available to ServiceMix by adding its URL to the org.ops4j.pax.url.mvn.repositories property in etc/org.ops4j.pax.url.mvn.cfg
  - Could be a local file system directory or a repository manager that you import the archive into.

Let’s look at the example...

Rider Auto Parts: Problem to be Solved

- Frontend receives messages from web store via SOAP/HTTP and FTP
- Message payloads can be CSV or XML from the FTP
- Backend service needs POJO payload
- There is a no downtime requirement when replacing backend

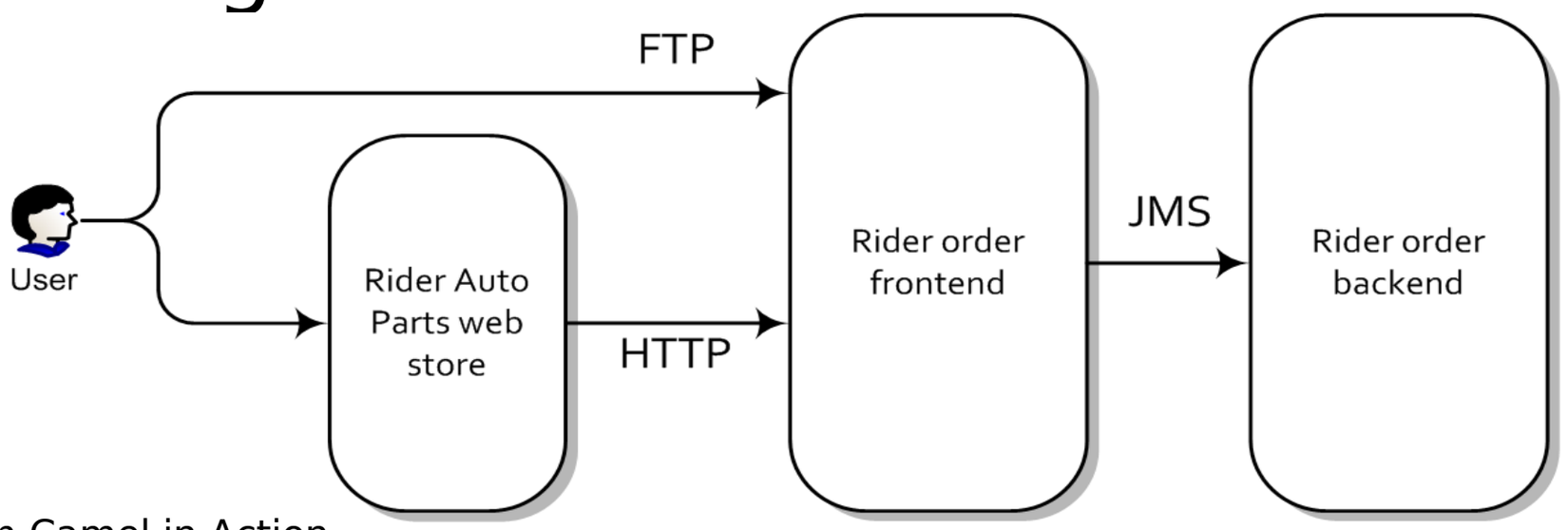


Diagram from Camel in Action.

Rider Auto Parts: Implementation Details

- Apache ServiceMix 4.5.1 used as the deployment container
- Each subsystem deployed as OSGi bundle
- Camel used as the integration framework
- CXF used to provide web service support
- Leverage ActiveMQ to decouple subsystems

Rider Auto Parts: Deployment Architecture

CamelOne

Demo time!

Useful references

- Apache ServiceMix - http://servicemix.apache.org
- Apache Camel - http://camel.apache.org
- Fuse IDE - http://fusesource.com/products/fuse-ide
- Camel in Action - http://manning.com/ibsen
- Example source - https://github.com/janstey/rider-auto-osgi/tree/servicemix-4.5